Adaptive Multicore Scheduling for the LTE Uplink

Maxime Pelcat, Jean François Nezan, Slaheddine Aridhi

Adaptive Multicore Scheduling for the LTE Uplink

Maxime Pelcat, Jean-Francois Nezan
IETR, INSA Rennes, CNRS UMR 6164, UEB,
20, Av. des Buttes de Coesmes,
F 35708 Rennes Cedex 7,
{mpelcat, jnezan}@insa-rennes.fr

Slaheddine Aridhi
Texas Instruments, HPMP Division,
06271 Villeneuve Loubet, France,
saridhi@ti.com

Abstract

The next generation cellular system of 3GPP is named Long Term Evolution (LTE). Each millisecond, a LTE base station receives information from up to one hundred users. Multicore heterogeneous embedded systems with Digital Signal Processors (DSP) and coprocessors are power efficient solutions to decode the LTE uplink signals in base stations. The LTE uplink is a highly variable algorithm. Its multicore scheduling must be adapted every millisecond to the number of connected users and to the data rate they require.

To solve the issue of the dynamic deployment while maintaining low latency, one approach would be to find efficient on-the-fly solutions using techniques such as graph generation and scheduling. This approach is opposed to a static scheduling of predefined cases. We show that the static approach is not suitable for the LTE uplink and that present DSP cores are powerful enough to recompute an efficient adaptive schedule for the LTE uplink most complex cases in real-time.

1 Introduction

The 3rd Generation Partnership Project (3GPP) telecommunication standards, namely GSM and 3G, are now used by billions of people over the world. The new 3GPP Long Term Evolution telecommunication standard (LTE) enables peak downlink data rates over 100Mbit/s, peak uplink data rates over 50Mbit/s, cells with radius over 100km and reliability in unfavorable transmission conditions. Moreover, up to one hundred users can share simultaneously the available bandwidth. This high performance is enabled by advanced techniques including Multiple Input Multiple Output (MIMO), Orthogonal Frequency-Division Multiple Access (OFDMA) and Single-Carrier Frequency-Division Multiple Access (SC-FDMA) [5] [16]. These techniques must be implemented in the Open System Interconnection (OSI) Layer 1, also named physical layer, of the LTE base stations (called evolved NodeB or eNodeB) and User Equipments (UEs). They come at the expense of computational power in both eNodeBs and UEs. The processor frequencies are nowadays limited in order to limit system power consumption. One power efficient solution to implement complex algorithms in real-time is to use multicore Digital Signal Processor (DSP) platforms with hardware coprocessors. This paper deals with the efficient implementation of the LTE physical layer onto such heterogeneous embedded systems.

Decoding the LTE uplink physical layer in the eNodeB consists in receiving the multiplexed data from connected UEs, decoding it and transmitting it to the upper layers of the standard. Depending on the number of active UEs and on their instant data rate, the decoding load varies dramatically. We propose in this paper a multicore adaptive scheduler with the capacity to recompute every millisecond the multicore mapping of the algorithm. It uses dataflow graph transformation and scheduling techniques, usually executed offline. The adaptive scheduler is in charge of keeping a low uplink completion time. LTE has strict constraints in terms of response time, limiting the available time for uplink and downlink. The scheduler disposes of a graph modeling the execution and the Deterministic-Actor-Execution-Time (DAET) of each function, i.e. the time needed to execute each function on each processing element (core or coprocessor). The mapping is chosen after an online simulation of the application execution.

The problem of LTE uplink scheduling is described in Section 2 and a model of the algorithm is proposed. Section 3 explains the limits of precalculated schedules, gives some related works and presents the adaptive scheduler. Experimental results are given in Section 4 that show the suitability of the technique in terms of computational power, memory and schedule quality, even in the LTE most complex cases. They prove that new DSP hardware architectures including several powerful cores bring new multicore scheduling challenges as well as an enhanced processing

power to solve them.

2 Implementing the LTE Uplink Physical Layer onto a Multicore Architecture

In this Section, we analyze the problem of LTE uplink decoding and divide it into a static and a dynamic part.

2.1 The Uplink Decoding

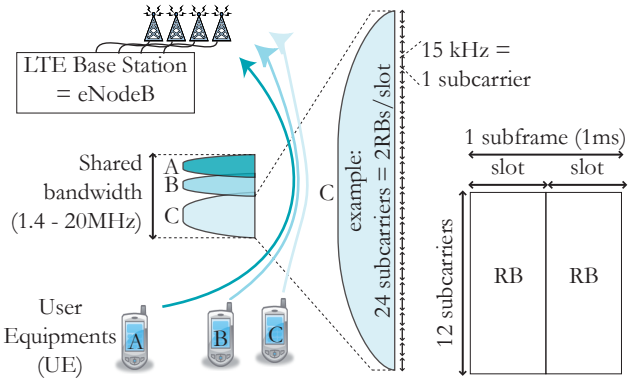

Figure 1. The LTE uplink transmission.

The LTE uplink is symbolized in Figure 1. Each UE sends data to a base station in its own preallocated bandwidth. The technology used to multiplex the UE pieces of data is SC-FDMA [16]. It can possibly be enhanced by MIMO, as the UEs and eNodeBs have respectively up to 2 and 4 antennas. The available bandwidth is divided into subcarriers separated by 15 kHz. The eNodeB Medium Access Control (MAC) scheduler is a decision module of the OSI layer 2 that chooses how to distribute the resources between UEs. It assigns resource elements of 180 kHz (12 sub-carriers) by 1ms (a subframe) to the connected UEs. Such an element contains a couple of Resource Blocks (RB) as shown in Figure 1.

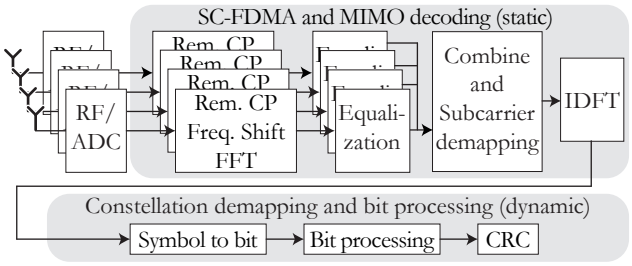

Figure 2. The uplink decoding static and dynamic parts.

The physical layer decoding operation can be roughly divided in two parts as shown in Figure 2 :

1. The static SC-FDMA and MIMO decoding. The parameters of this part are fixed during runtime (number of receiving antennas, Frequency Division Duplex (FDD) or Time Division Duplex (TDD), chosen bandwidth...). It removes the Cyclic Prefix (CP), shifts frequency to compensate the Doppler effect due to UE movements, converts the symbols into frequency domain (FFT) and equalizes them using received reference signals. The data from up to 4 antennas are then combined and the subcarriers reordered. Finally, an IDFT brings back the data into time domain.

2. The dynamic constellation demapping and bit processing. The parameters of this part are highly variable at runtime: number of connected UEs, number of allocated Resource Blocks, symbol constellation... The multicore scheduling of this dynamic part must be adaptive. A graph description of this part will be detailed in Section 2.3.

The first static part can be represented as a Synchronous Dataflow graph (SDF [12]) and is quite parallel. Consequently, a rapid prototyping tool like PREESM [15] can efficiently parallelize it at compile time. The multicore scheduling of the second dynamic part needs to be adapted to the varying parameters. The constraints on these varying parameters are detailed in the next Section.

2.2 Application Constraints

Figure 3 shows the different bandwidth configuration of the LTE physical layer. While the bandwidth varies between 1.4 and 20 MHz, the number of resource blocks in one slot $N_{RB_{MAX}}$ varies between 6 and 100. The larger the bandwidth is, the more manifold allocation cases are and the more complex the adaptive scheduling becomes.

BW (MHz)	1.4	3	5	10	15	20
Data Subcarriers	72	144	300	600	900	1200
Resource Blocks	6	12	25	50	75	100

Figure 3. The LTE bandwidth configurations.

In the 20MHz case, the MAC scheduler assigns every millisecond up to 100 couples of Resource Blocks to a maximum of 100 UEs. Depending on its choices, the uplink physical layer dynamic part needs to be rescheduled onto the multicore architecture. Next Section explains the model used to represent the uplink physical layer dynamic part.

2.3 Dataflow Description of the LTE Uplink Dynamic part

In order to schedule every millisecond the decoding of one LTE uplink subframe, we need a parameterized model of the algorithm. The chosen model is a dataflow graph because dataflow graphs (SDF [12], CSDF for Cyclo-Static Dataflow [13]...) have proven to be efficient representations for signal processing applications, in particular due to the Ptolemy II project [11]. The vertices of dataflow graphs are called actors. They consume data on their input edges, process them and produce the results on their output edges. They exchange data exclusively via edges. An actor is called (or fired) as long as there is data to consume on its input edges. Actors can migrate from one processing element to another with no side effects on shared data.

We call the graph model used to represent the dynamic part of the LTE uplink physical layer the Parameterized Cyclo-Static Directed Acyclic Graph (PCSDAG). It is a subset of the CSDF model. It presents two main advantages:

- As a subset of CSDF, it can compactly model an algorithm with complex production or consumption patterns. It is the case of our algorithm because the number of resource blocks assigned to each user varies much.
- The simplifications enable an fast transformation into an Homogeneous Directed Acyclic Graph (HDAG) ready to be scheduled.

The simplifications comparing to CSDF are that:

- a PCSDAG has no cycle,
- it contains only one vertex without input edge,
- the number of firings of this first actor is fixed to 1.

An HDAG is also a graph with no cycle and in which each edge have equal data production and consumption. The HDAG actors are instances of PCSDAG actors. Each HDAG actor is fired only once.

The data token production and consumption of each PCSDAG edge are set by the MAC scheduler. They can either be a single integer value or a pattern of integer values. For example, a consumption of the number of resource blocks per UE can contain the pattern $\{10, 5, 3, 1, 1\}$, meaning that the actor will consume 10 data on the first firing, 5 on the second and so on. 1 millisecond before each subframe, all the production and consumption patterns are set by the MAC scheduler. We can then expand the PCSDAG into an HDAG and schedule the HDAG.

Figure 4 shows a simplified view of the LTE uplink PCSDAG description. The shape of the graph depends on the following parameters:

- the current number of UEs: nb_UE,
- A pattern of parameters giving the number of couples of resource blocks allocated in each subframe to each UE: $RBs\_UE = \{RBs\_UE1, RBs\_UE2, RBs\_UE3...\}$. This pattern has a maximum length of 100.
- the maximum number of resource blocks allocated in each slot to one UE: max_RBs_UE.

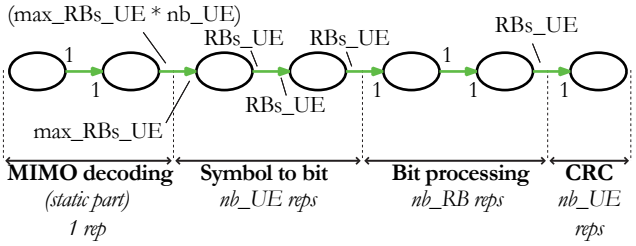

Figure 4. PCSDAG description of the LTE uplink dynamic part with data productions and consumptions recomputed every millisecond.

The Deterministic-Actor-Execution-Times of the LTE uplink graph actors also depend on parameters set by the MAC scheduler. The MIMO decoding part in Figure 4 correspond to the expanded static part in Figure 2. The next vertices successively convert the symbols in bits and then process those bits to prepare a data frame for the OSI Layer 2.

2.4 Target Architectures

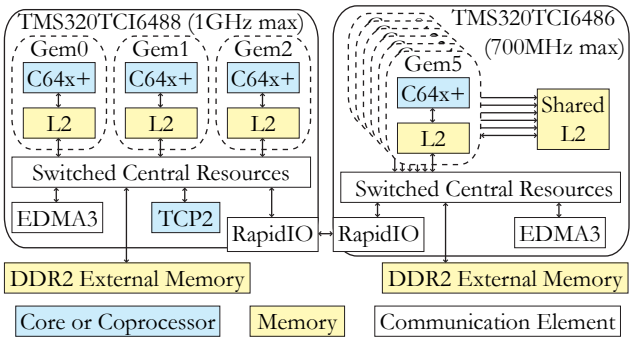

Figure 5. Target architecture example: two multicore DSPs connected with a RapidIO serial link.

The architectures targeted by this study are the Texas Instruments high performance architectures based on the

c64x+ core. The 6-core TMS320TCI6486 DSP (tci6486) and the 3-core TMS320TCI6488 DSP (tci6488) are examples of such architectures. Any combination these DSPs (interconnected with RapidIO [2] serial links for instance) are also targeted architectures. The dataflow graph scheduling techniques naturally handle heterogeneity in data links and processing element, enabling targets with coprocessors, different DSP frequencies and different communication media. The tci6488, for instance, includes a turbo decoding coprocessor that can help decoding the uplink physical layer. Figure 5 gives a simplified view of an architecture example with a tci6486 connected to a tci6488 by a RapidIO link. The tci6488 has only distributed on-chip memory, local to each c64x+ core, while the tci6486 also includes a shared on-chip memory of 768 KBytes. They both contain an Enhanced Direct Memory Access (EDMA [7]) to transfer data in parallel with calculation.

3 Adaptive Multicore Scheduling of the LTE Uplink Physical Layer

An obvious solution to efficiently schedule a variable algorithm onto such a heterogeneous architecture is to schedule it offline in all its configurations and switch between the pre-computed schedules online. We will now analyze the limits of such an approach.

3.1 The Limits of Pre-computed Scheduling

Assigning a given number of resource blocks to UEs is a problem equivalent to partitioning the number of resource blocks into a sum of integers. The problem of integer partition is illustrated by Ferrer diagrams in Figure 6. Given a number N_{RB} of resource blocks to assign, it gives the number of different allocation configurations $p(N_{RB})$, i.e. the possible number of different graphs to schedule and map for a given amount of resource blocks. The number of resource blocks N_{RB} allocated every millisecond also varies between 0 and $N_{RB_{MAX}}$. In Section 2.2, $N_{RB_{MAX}}$ was shown to be a constant for each eNodeB fixed between 6 and 100 depending on the LTE uplink bandwidth. The total number of different graphs for a given eNodeB uplink is:

$$P(N_{RB_{MAX}}) = \sum_{i=1}^{N_{RB_{MAX}}} p(i). \quad (1)$$

For the simplest case of a 1.4MHz uplink bandwidth with only 6 RBs to allocate, the number of graphs is 29. It is possible to precompute and store the scheduling of these 29 graphs. However, the number of graphs increases exponentially with N_{RB} and the 3MHz case with 12 RBs already generates 271 cases. As a consequence, the scheduling of all the LTE uplink cases with bandwidth higher than 3MHz can not be statically scheduled. As an example, $P(50) = 1.295.970$.

Many papers have detailed multicore scheduling of static applications onto heterogeneous architectures. Next Section will explain the differences between some related works and the current study.

3.2 Some Related Works on Dataflow Scheduling

The multicore scheduling of dataflow actors is composed of two operations:

- Mapping consists in choosing a processing element to execute each actor.
- Scheduling consists in finding an order of execution for all the actors mapped on a given processing element.

Both operations must be executed jointly for the multicore scheduling decisions to be appropriate. The main problem in LTE uplink scheduling is the fast variability of the graph size and shape over time. In [9], quasi-static scheduling techniques are cited and developed that schedule dynamic graphs at compile time. All possible combinations of parameters are scheduled, which would be impossible here because the number of cases is very high (see Section 3.1). These techniques are more adapted to Boolean conditions than to parameters that can switch anytime between as much as 100 possibilities. This problem justifies the use of online scheduling instead of quasi-static offline scheduling.

Another dataflow online scheduling method is developed in [4]. However, it focuses on image processing algorithms that can be modeled by a flow-shop problem, i.e. several independent jobs each composed of sequential actors. The LTE uplink can not be modeled that way. The technique is thus not applicable here.

In [10], many static scheduling techniques are tested and some efficient ones proposed. We use, in this paper, a simplified version of the list scheduler described in [10]. It shows efficiency for the current application scheduling and stays feasible with strict timing scheduling time constraints. The adaptive scheduler, presented in next Section, is the combination of a graph expansion and this list scheduler.

3.3 Structure of the adaptive Scheduler

The adaptive multicore scheduler is illustrated in Figure 7. It contains an initialization phase that generates the PCSDAG and precomputes the graph parameters expressions. The rest of the processing is called each millisecond when the MAC scheduler changes the resource block allocation. The scheduler consists in 2 steps:

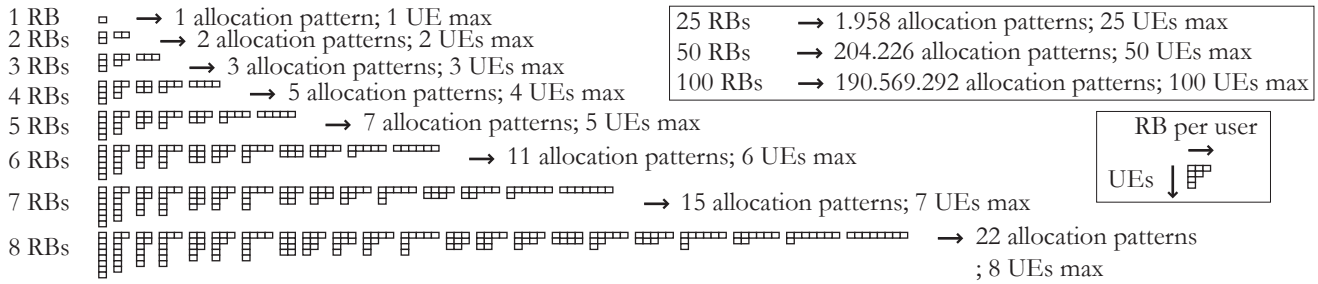

Figure 6. The problem of allocating resource blocks to UEs is equivalent to integer partitions represented here in Ferrer diagrams.

- The graph expansion step transforms the PCSDAG into a HDAG with a shape depending on the PCSDAG parameters.
- The list scheduling maps each actor in the graph to a processing element (core or coprocessor) in the architecture.

The architecture model is a simplified version of the one in [14]. It specifies the data rates between each pair of processing elements, mapping constraints and DAET, i.e. which actors can be executed by each processing element and the time necessary for each computation. The goal of the initialization step is to reduce as much as possible the loop complexity. It parses the expressions of the PCSDAG edges productions and consumptions and converts them into a Reverse Polish Notation (RPN) stack using the Shunting yard algorithm [6]. RPN specifies in a bracket-less expression the order of expression evaluation.

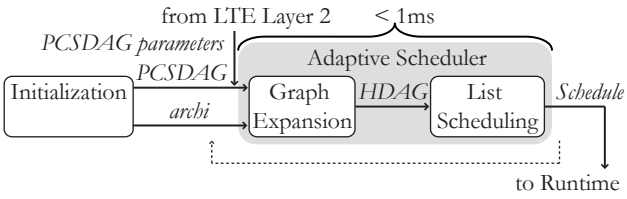

Figure 7. Adaptive multicore scheduling steps: graph expansion and list scheduling are called in a loop every millisecond.

The output of the adaptive scheduler feeds a runtime. This runtime can for instance implement a multicore API (OpenCL [1], Multicore Association [3]...). The sum of graph expansion and list scheduling execution times must stay under 1ms for the scheduler to be usable in the LTE uplink. These two operations are detailed in next Sections.

3.4 Adaptive expansion of PCSDAG into Homogeneous Directed Acyclic Graph

Figure 8 gives 4 examples of HDAG graphs generated from expansion with different allocation schemes of the PCSDAG in Figure 4. The number of vertices in the HDAG is:

$$V = 2 + 3 * nb\_UE + 2 * N_{RB}. \quad (2)$$

The maximal size of the HDAG is thus 502 vertices. To generate the second case (2 UEs - 5 RBs), we have $nb\_UE = 2$, $RBs\_UE = \{2, 3\}$,and $max\_RBs\_UE = 3$ in the PCSDAG graph.

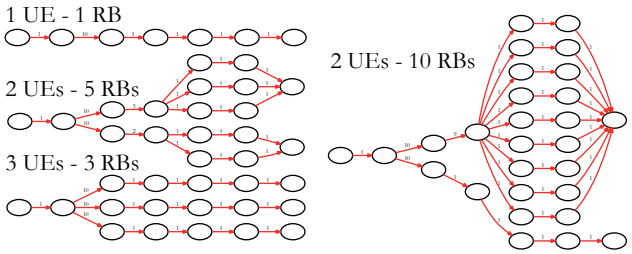

Figure 8. Examples of HDAG graphs generated from the PCSDAG description in Figure 4.

The PCSDAG graph model has been chosen to simplify and fasten the expansion step. The adaptive scheduler calculates the number of firings of each actor, gathered in the so-called Basis Repetition Vector (BRV [17]), using only the firings of the preceding actors and the productions and consumptions of the incoming edges. Productions and consumption are computed from RPN expression stacks. The fact that there is no loop in the graph and that there is only one source vector with a firing of 1 enables this greedy

method. There is no schedulability checking in the expansion step. The graph schedulability can be verified offline using rapid prototyping tools like PREESM [15]. Once the BRV has been calculated and the HDAG vertices have been created, the appropriate edges are added to interconnect them.

By comparison with SDF, SDF schedulability checking and expansion into HSDF requires to study the null space of an M-by-N matrix called topology matrix [17] with M the number of vertices and N the number of edges. This kind of operation is too costly to be executed in an online scheduler.

When the PCSDAG graph has been expanded, the resulting HDAG is ready for the list scheduling explained in the next Section.

3.5 List scheduling of the Homogeneous Directed Acyclic Graph

The list scheduling used is a simplified version of the greedy algorithm described in [10]. It is greedy in that it never questions one of its previous mapping choices. The list scheduling process is illustrated in Figure 9.

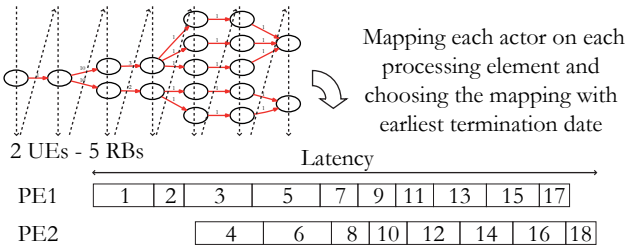

Figure 9. Example of the list scheduling of HDAG actors.

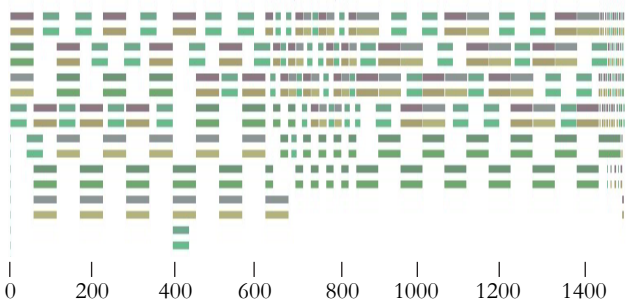

Figure 10. Example of a schedule Gantt chart: case with 100 RBs and 100 UEs on the architecture from Figure 5 with one core reserved for the scheduling.

There is no actor reordering process before executing the list scheduling. In the LTE uplink case, this has no consequence on the scheduling quality because all the paths from the first actor to the actors without successors are equivalent. In a more complex case, a suboptimal input list multiplies the completion time by a factor $\lambda \leq 2 - 1/n$ [8] where n is the number of target cores. This result is an approximation because the data transmission latencies are ignored.

The next Section gives some implementation results of the adaptive scheduler.

4 Implementation and Experimental results

A special care is given to the adaptive scheduler implementation genericity, compactness and speed. The whole code is written in C++ with private members and inlined accessors.

4.1 Memory footprint of the adaptive scheduler

In order to reduce the execution time, the adaptive scheduler contains no dynamic allocation. Its memory footprint (Figure 11) is only 126 KBytes. Each c64x+ core of the tci6486 and tci6488 processors presented in Section 2.4 respectively has an internal local memory of 608 KBytes and 500 to 1500 KBytes. The footprint is small enough to fit in the internal local L2 memory of any of these cores.

Half of the memory footprint is taken by the graphs. The PCSDAG great size is due to the variable patterns stored in RPN. The HDAG graph has a sufficient size to contain any LTE uplink configuration. One third of the footprint is taken by the code. Compilation is optimized for speed; the code size could be reduced by optimizing for code size.

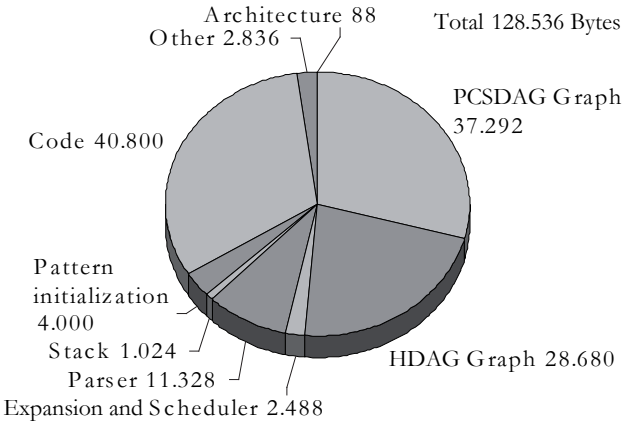

Figure 11. Memory footprint, in Bytes, of the statically allocated adaptive scheduler.

4.2 Impact of the Graph and Architecture Sizes on the Adaptive Scheduler Speed

Figure 12 shows that the execution time increases linearly when the graph size increases. The graph expansion time for very small HDAGs is due to the complex RPN variables evaluation. The worst case execution time is less than 950.000 cycles, enabling real-time on a c64x+ at 1GHz. One core of a tci6488 can thus reach real-time with this code.

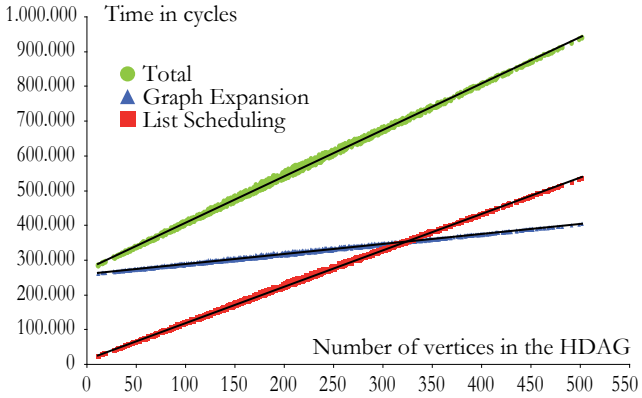

Figure 12. Impact of the HDAG graph size on the execution time of the LTE uplink scheduling on the architecture in Figure 5 with one core reserved for the scheduling.

In Figure 13, we can see that increasing the number of cores of the architecture also augments the adaptive scheduler execution time linearly. The maximum number of target cores with the present implementation and a DSP at 1GHz is 9 (including one for scheduling). Specific Texas Instruments optimizations with intrinsic and pragma can improve this result.

4.3 Evaluating the Limits of the Target Architecture

Depending on the current number of decoded RBs and the current number of communicating UEs, the completion time of the LTE uplink dynamic part changes. Figure 14 shows the speedup obtained due to the use of multicore instead of a single c64x+ at 700MHz. It approaches the theoretical maximum of 8.8 as soon as the graph becomes big enough. We can see in Figure 15 that the architecture in Figure 5 with one core of the tci6488 kept for scheduling and no coprocessor is sufficient to decode the LTE uplink in 1 millisecond for 50 RBs and 50 UEs, i.e. for the 10MHz bandwidth case. The static part of the algorithm is not taken

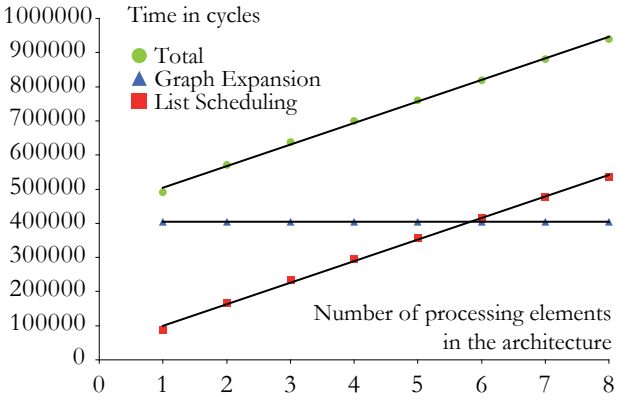

Figure 13. Impact of the number of processing elements on the LTE uplink scheduling execution time with a LTE uplink worst case.

into account in this simulation. The problem of the multicore partitioning of the static part is easier because it can be solved offline. It will need to be pipelined with the dynamic part in order to respect the 1ms execution time limit.

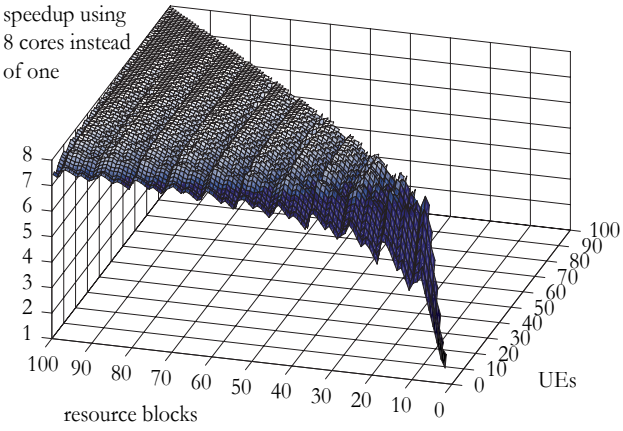

Figure 14. Schedule speedup vs. number of RBs and UEs using the architecture from Figure 5.

In order to solve the 20MHz case, some coprocessors must be used or some cores added. As long as the number of processing elements is kept under 9, the adaptive scheduler executed on a c64x+ at 1GHz can solve the 20MHz problem as is.

5 Conclusion

In this paper, we present a dataflow scheduling algorithm applied to the LTE uplink physical layer decoding and

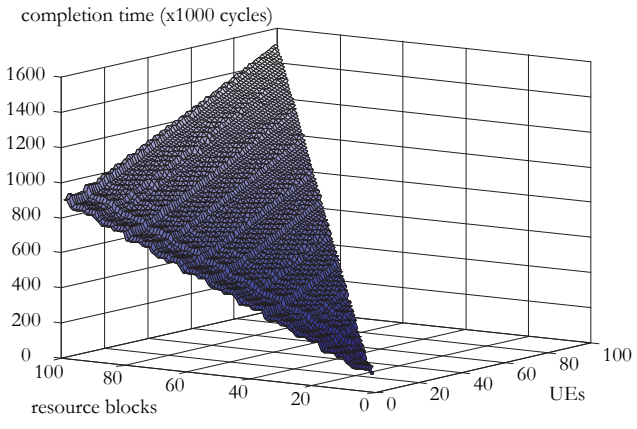

Figure 15. Impact of the number of RBs and UEs on the completion time of the uplink dynamic part on the architecture in Figure 5; the limit of 1ms is reached for a maximum of 50 RBs and 50 UEs, i.e. the 10MHz case.

executed on a heterogeneous multicore architecture. The scheduling is executed on-line to adapt the multicore mapping choices to the application variations. We show that off-line solutions are not suited for the current application. Moreover, results prove that the scheduling execution time and the results of the scheduling are compatible with the strict time constraints of LTE.

The method is not limited to the LTE uplink algorithm. The adaptive multicore scheduler targets any highly variable algorithm that can be described with a Parameterized Cyclo-Static Directed Acyclic Graph, including the LTE downlink encoding.

References

[1] OpenCL. http://www.khronos.org/opencl/.

[2] RapidIO. http://www.rapidio.org/home/.

[3] The Multicore Association. http://www.multicore-association.org/home.php.

[4] J. Boutellier, S. S. Bhattacharyya, and O. Silvn. A low-overhead scheduling methodology for fine-grained acceleration of signal processing systems. *Journal of Signal Processing Systems*, 2009.

[5] E. Dahlman, S. Parkvall, J. Skold, and P. Beming. *3G Evolution: HSPA and LTE for Mobile Broadband*. Academic Press Inc, June 2007.

[6] E. Dijkstra. Algol 60 translation. *Supplement, Algol 60 Bulletin*, 10, 1960.

[7] B. Feng and R. Salman. TMS320TCI6482 EDMA3 performance, texas instrument technical document (SPRAAG8), Nov. 2006.

[8] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45, 1966.

[9] S. Ha and E. A. Lee. Compile-time scheduling of dynamic constructs in dataflow program graphs. *IEEE Transactions on Computers*, 46, 1997.

[10] Y. Kwok. *High-performance algorithms of compile-time scheduling of parallel processors*. PhD thesis, Hong Kong University of Science and Technology (People's Republic of China), 1997.

[11] E. Lee. Overview of the ptolemy project. *Technical memorandum UCB/ERL M01/11, University of California at Berkeley*, 2001.

[12] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on computers*, 36(1), 1987.

[13] T. M. Parks, J. L. Pino, and E. A. Lee. A comparison of synchronous and cyclo-static dataflow. In *Proc. of ASILOMAR*, page 204210, 1995.

[14] M. Pelcat, J. F. Nezan, J. Piat, J. Croizer, and S. Aridhi. A System-Level architecture model for rapid prototyping of heterogeneous multicore embedded systems, 2009.

[15] M. Pelcat, J. Piat, M. Wipliez, J. F. Nezan, and S. Aridhi. An open framework for rapid prototyping of signal processing applications. *EURASIP Journal on Embedded Systems*, 2009.

[16] S. Sesia, I. Toufik, and M. Baker. *LTE, The UMTS Long Term Evolution: From Theory to Practice*. Wiley, 2009.

[17] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC, 1 edition, Mar. 2000.